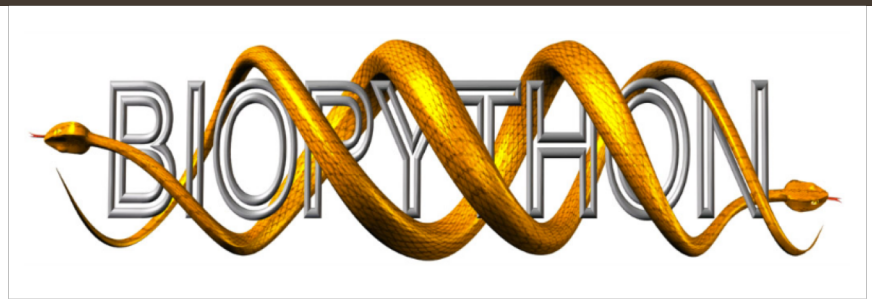
université de BORDEAUX

Formation CNRS
8 Novembre 2018
Python pour la biologie

Biopython



cgfb
BIOINFORMATIQUE

université de BORDEAUX

Parsing sequences file formats : L'objet SeqRecord

- Bioinformatics work involves dealing with the many types of file formats designed to hold biological data.
- These files are loaded with interesting biological data, and a special challenge is parsing these files into a format so that you can manipulate them with some kind of programming language.
- However, the task of parsing these files can be frustrated by the fact that the formats can change quite regularly, and that formats may contain small subtleties which can break even the most well designed parsers.
- Remember to load module SeqIO using
  - from Bio import SeqIO

Simple FASTA parsing example

➔ If you open the lady slipper orchids FASTA file ls_orchid.fasta (94 records) in your favourite text editor, you'll see that the le starts like this:

```
>gi|2765658|emb|Z78533.1|CIZ78533 C.irapeanum 5.8S rRNA gene and ITS1 and ITS2 DNA
CGTAACAAGGTTTCCGTAGGTGAACCTGCGGAAGGATCATTGATGAGACCGTGGAATAAACGATCGAGT
GAATCCGGAGGACCGGTGTACTCAGCTCACCGGGGGCATTGCTCCCGTGGTGACCCTGATTTGTTGTT
GGG
```

```
>>>from Bio import SeqIO
>>>for seq_record in SeqIO.parse("ls_orchid.fasta", "fasta"):
        print(seq_record.id)
        print(repr(seq_record.seq))
        print(len(seq_record))
gi|2765658|emb|Z78533.1|CIZ78533
Seq('CGTAACAAGGTTTCCGTAGGTGAACCTGCGGAAGGATCATTGATGAGACCGTGG...CGC', SingleLetterAlphabet())
740
[…]
gi|2765564|emb|Z78439.1|PBZ78439
Seq('CATTGTTGAGATCACATAATAATTGATCGAGTTAATCTGGAGGATCTGTTTACT...GCC', SingleLetterAlphabet())
592
```

Simple genbank parsing example

```
>>> for seq_record in SeqIO.parse("ls_orchid.gbk", genbank"):
        print(seq_record.id)
        print(repr(seq_record.seq))
        print(len(seq_record))
Z78533.1
Seq('CGTAACAAGGTTTCCGTAGGTGAACCTGCGGAAGGATCATTGATGAGACCGTGG...CGC', IUPACAmbiguousDNA())
740
[…]
Z78439.1
Seq('CATTGTTGAGATCACATAATAATTGATCGAGTTAATCTGGAGGATCTGTTTACT...GCC', IUPACAmbiguousDNA())
592
```

➔ Using Python iterator is within a list comprehension (or a generator expression)

```
>>> identifiers = [seq_record.id for seq_record in SeqIO.parse("ls_orchid.gbk", "genbank")]
>>> identifiers
['Z78533.1', 'Z78532.1', 'Z78531.1', 'Z78530.1', 'Z78529.1', 'Z78527.1', ..., 'Z78439.1']
```

➔ Also “swiss” for SwissProt files or “embl” for EMBL text files

➔ See wiki page (http://biopython.org/wiki/SeqIO)

Iterating over the records in a sequence file

- The object returned by Bio.SeqIO is actually an iterator which returns SeqRecord objects

```
>>>record_iterator = SeqIO.parse("ls_orchid.fasta", "fasta”)
>>>first_record = next(record_iterator)
>>>print(first_record.id)
>>>print(first_record.description)
```

Z78533.1

```
>>>second_record = next(record_iterator)
>>>print(second_record.id)
>>>print(second_record.description)
```

Z78439.1

```
>>>first_record = next(SeqIO.parse("ls_orchid.gbk", "genbank"))
```

Getting a list of the records in a sequence file

- Access records in any order using Python list data type Using a list
- Much more flexible than an iterator (length of the list) but need more memory (hold all the records in memory at once).

```
>>>records = list(SeqIO.parse("ls_orchid.gbk", "genbank"))
>>>print("Found %i records" % len(records))
```

```
Found 94 records
```

```
>>>last_record = records[-1] #using Python's list tricks
>>>print(last_record.id)
>>>print(repr(last_record.seq))
>>>print(len(last_record))
```

```
Z78439.1
Seq('CATTGTTGAGATCACATAATAATTGATCGAGTTAATCTGGAGGATCTGTTTACT...GCC', IUPACAmbiguousDNA())
592
```

```
>>>first_record = records[0] #remember, Python counts from zero
>>>print(first_record.id)
>>>print(repr(first_record.seq))
>>>print(len(first_record))
```

```
Z78533.1
Seq('CGTAACAAGGTTTCCGTAGGTGAACCTGCGGAAGGATCATTGATGAGACCGTGG...CGC',
IUPACAmbiguousDNA())
740
```

Extracting data

- As an example of how annotations are stored, we'll look at the output from parsing the first record in the GenBank file ls_orchid.gbk.
- Human readable summary of most of the annotation data for the SeqRecord

```
record_iterator = SeqIO.parse("ls_orchid.gbk", "genbank")
first_record = next(record_iterator)
print(first_record)
```

```
ID: Z78533.1
Name: Z78533
Description: C.irapeanum 5.8S rRNA gene and ITS1 and ITS2 DNA.
Number of features: 5
/sequence_version=1
/source=Cypripedium irapeanum
/taxonomy=['Eukaryota', 'Viridiplantae', 'Streptophyta', ..., 'Cypripedium']
/keywords=['5.8S ribosomal RNA', '5.8S rRNA gene', ..., 'ITS1', 'ITS2']
/references=[...]
/accessions=['Z78533']
/data_file_division=PLN
/date=30-NOV-2006
/organism=Cypripedium irapeanum
/gi=2765658
Seq('CGTAACAAGGTTTCCGTAGGTGAACCTGCGGAAGGATCATTGATGAGACCGTGG...CGC', IUPACAmbiguousDNA())
```

- .annotations attribute which is just a Python dictionary. Like any Python dictionary, you can easily get a list of the keys and values:

```
print(first_record.annotations)
print(first_record.annotations.keys())
print(first_record.annotations.values())
```

- extract a list of the species from the ls orchid.gbk GenBank le. The information we want is held in the annotations dictionary under `source' and `organism':

```
>>> print(first_record.annotations["source"])
Cypripedium irapeanum
```

```
>>> print(first_record.annotations["organism"])
Cypripedium irapeanum
```

- In general, `organism' is used for the scientific name (in Latin, e.g. Arabidopsis thaliana), while `source' will often be the common name (e.g. thale cress)

- In general, `organism' is used for the scientific name (in Latin, e.g. Arabidopsis thaliana), while `source' will often be the common name (e.g. thale cress)

```
from Bio import SeqIO
all_species = []
for seq_record in SeqIO.parse("ls_orchid.gbk", "genbank"):
all_species.append(seq_record.annotations["organism"])
print(all_species)
```

```
from Bio import SeqIO
all_species = [seq_record.annotations["organism"] for seq_record in \
SeqIO.parse("ls_orchid.gbk", "genbank")]
print(all_species)
```

```
['Cypripedium irapeanum', 'Cypripedium californicum', ..., 'Paphiopedilum barbatum']
```

Filtering a sequence file

- large file with many sequences in it (e.g. FASTA le or genes, or a FASTQ or SFF file of reads)

```
from Bio import SeqIO
input_file = "big_file.sff"
id_file = "short_list.txt"
output_file = "short_list.sff"
wanted = set(line.rstrip("\n").split(None,1)[0] for line in open(id_file))
print("Found %i unique identifiers in %s" % (len(wanted), id_file))
records = (r for r in SeqIO.parse(input_file, "sff") if r.id in wanted)
count = SeqIO.write(records, output_file, "sff")
print("Saved %i records from %s to %s" % (count, input_file, output_file))
if count < len(wanted):
print("Warning %i IDs not found in %s" % (len(wanted)-count, input_file))
```

- Let's say the list of IDs is in a simple text file, as the first word on each line.
- This could be a tabular file where the first column is the ID
- Note that we use a Python set rather than a list, this makes testing membership faster

Producing randomised genomes

- Same search on randomised versions of the same genome for statistical analysis
- get it from our website, NC_005816.gb
- read it in as a SeqRecord using the Bio.SeqIO.read() function

```
>>> from Bio import SeqIO
>>> original_rec = SeqIO.read("NC_005816.gb", "genbank")
```

- Python random module, in particular the function random.shuffle

```
>>> import random
>>> nuc_list = list(original_rec.seq)
>>> random.shuffle(nuc_list) #acts in situ!
```

```
>>> from Bio.Seq import Seq
>>> from Bio.SeqRecord import SeqRecord
>>> shuffled_rec = SeqRecord(Seq("".join(nuc_list), original_rec.seq.alphabet),
... id="Shuffled", description="Based on %s" % original_rec.id)
```

- Construct a new SeqRecord with a new Seq object using this shuffled list.

Complete script (version 1)

```
import random
from Bio.Seq import Seq
from Bio.SeqRecord import SeqRecord
from Bio import SeqIO
original_rec = SeqIO.read("NC_005816.gb","genbank")
handle = open("shuffled.fasta", "w")
for i in range(30):
nuc_list = list(original_rec.seq)
random.shuffle(nuc_list)
shuffled_rec = SeqRecord(Seq("".join(nuc_list), original_rec.seq.alphabet), \
id="Shuffled%i" % (i+1), \
description="Based on %s" % original_rec.id)
handle.write(shuffled_rec.format("fasta"))
handle.close()
```

Complete script (version 2)

```
import random
from Bio.Seq import Seq
from Bio.SeqRecord import SeqRecord
from Bio import SeqIO
def make_shuffle_record(record, new_id):
nuc_list = list(record.seq)
random.shuffle(nuc_list)
return SeqRecord(Seq("".join(nuc_list), record.seq.alphabet), \
id=new_id, description="Based on %s" % original_rec.id)
original_rec = SeqIO.read("NC_005816.gb","genbank")
shuffled_recs = (make_shuffle_record(original_rec, "Shuffled%i" % (i+1)) \
for i in range(30))
handle = open("shuffled.fasta", "w")
SeqIO.write(shuffled_recs, handle, "fasta")
handle.close()
```

Translating a FASTA file of CDS entries

```
from Bio.SeqRecord import SeqRecord
def make_protein_record(nuc_record):
"""Returns a new SeqRecord with the translated sequence (default table)."""
return SeqRecord(seq = nuc_record.seq.translate(cds=True), \
id = "trans_" + nuc_record.id, \
description = "translation of CDS, using default table")
```

```
from Bio import SeqIO
proteins = (make_protein_record(nuc_rec) for nuc_rec in \
SeqIO.parse("coding_sequences.fasta", "fasta"))
SeqIO.write(proteins, "translations.fasta", "fasta")
```

```
>>>first_record = records[0] #remember, Python counts from zero
>>>print(first_record.id)
>>>print(repr(first_record.seq))
>>>print(len(first_record))
```

```
Z78439.1
Seq('CATTGTTGAGATCACATAATAATTGATCGAGTTAATCTGGAGGATCTGTTTACT...GCC', IUPACAmbiguousDNA())
592
```

Simple quality filtering for FASTQ files

Trimming of primer sequences

Trimming off adaptor sequences

Converting FASTQ files

Converting FASTA and QUAL files into FASTQ files